

Symmetric Encryption

Thierry Sans

Design principles (reminder)

1. **Kerkoff Principle**

The security of a cryptosystem must not rely on keeping the algorithm secret

2. **Diffusion**

Mixing-up symbols

3. **Confusion**

Replacing a symbol with another

4. **Randomization**

Repeated encryptions of the same text are different

The attacker's model

- **Exhaustive Search**

Try all possible n keys (in average it takes $n/2$ tries)

- **Ciphertext only**

You know one or several random ciphertexts

- **Known plaintext**

You know one or several pairs of random plaintext and their corresponding ciphertexts

- **Chosen plaintext**

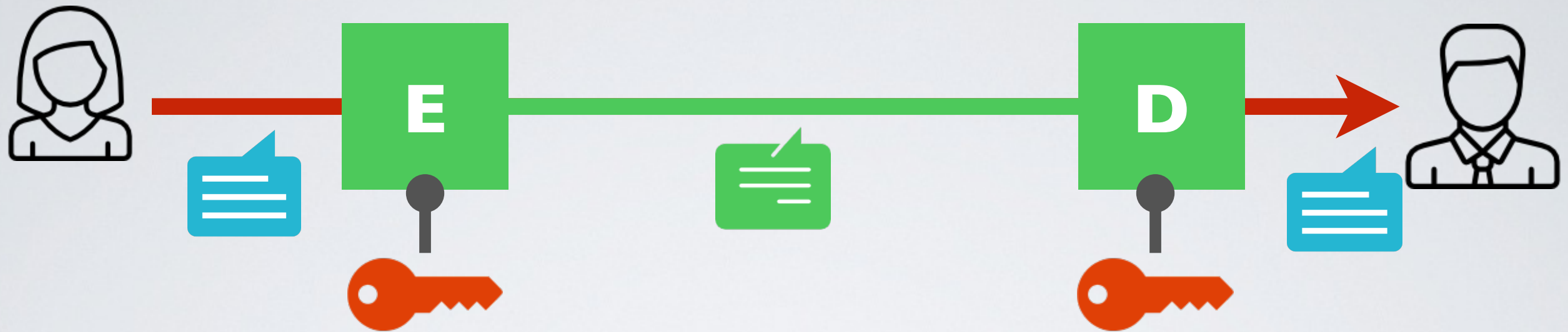
You know one or several pairs of chosen plaintext and their corresponding ciphertexts

- **Chosen ciphertext**

You know one or several pairs of plaintext and their corresponding chosen ciphertexts

➔ **A good crypto system resists all attacks**

Functional Requirements



➡ The same key k is used for encryption E and decryption D

1. $D_k(E_k(m))=m$ for every k , E_k is an injection with inverse D_k
2. $E_k(m)$ is easy to compute (either polynomial or linear)
3. $D_k(c)$ is easy to compute (either polynomial or linear)
4. $c = E_k(m)$ finding m is hard without k (exponential)

Outline

Stream cipher

RC4 - Rivest Cipher 4

Block cipher

- Encryption standards

DES (and 3DES) - Data Encryption Standard

AES - Advanced Encryption Standard

- Block cipher modes of operation

Stream Cipher

XOR Cipher (a.k.a Vernham Cipher)

a modern version of Vigenere

Use \oplus to combine the message and the key

$$E_k(m) = k \oplus m$$

$$D_k(c) = k \oplus c$$

Problem : known-plaintext attack

$$D_k(E_k(m)) = k \oplus (k \oplus m) = m$$

$$\text{so } k = (k \oplus m) \oplus m$$

$$x \oplus x = 0$$

$$x \oplus 0 = x$$

Mauborgne Cipher - a modern version of OTP

Use a random stream as encryption key

➡ Defeats the know-plaintext attack

Problem : Key-reused attack (a.k.a two-time pad)

$$C_1 = k \oplus m_1$$

$$C_2 = k \oplus m_2$$

$$\begin{aligned}\text{so } C_1 \oplus C_2 &= (k \oplus m_1) \oplus (k \oplus m_2) \\ &= (m_1 \oplus m_2) \oplus 0 \\ &= (m_1 \oplus m_2)\end{aligned}$$

| |
|------------------|
| $x \oplus x = 0$ |
| $x \oplus 0 = x$ |

Random Number Generator

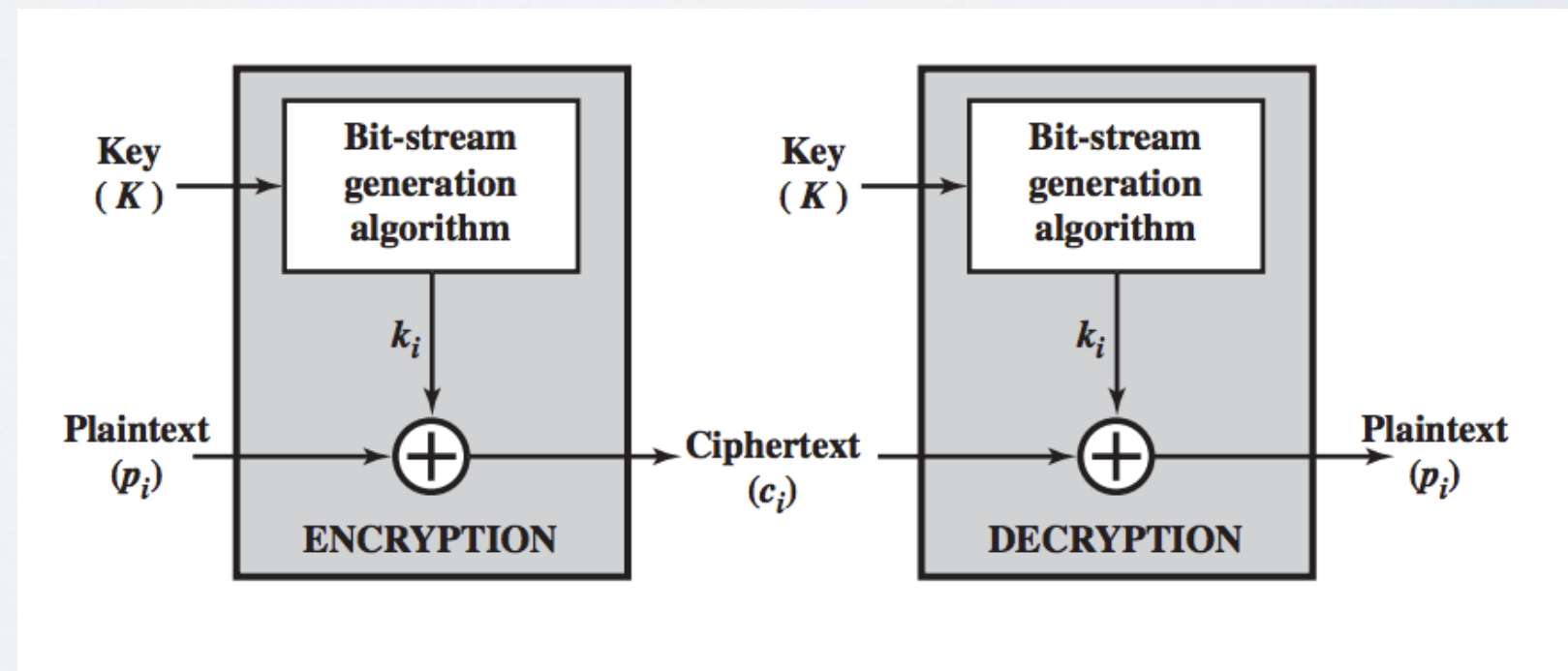
```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

True Random Number Generator

➔ No, because we want to be able to encrypt and decrypt

Pseudo-Random Generator

➔ Stretch a fixed-size seed to obtain an unbounded random sequence



Stream cipher

Can we use k as a seed?

$$E_k(m) = m \oplus \text{RNG}(k)$$

➡ Be careful of key reused attack !

Typical usage : choose a new s and send it using another encryption scheme E'

$$E_k(m) = (E'_k(s) , m \oplus \text{RNG}(s))$$

RC4 - Rivest Cipher 4

| | |
|----------|-------------------|
| Key Size | 40 - 2048 bits |
| Speed | ~ 8 cycles / byte |

Very simple implementation

[Home](#) / [Business Software](#)

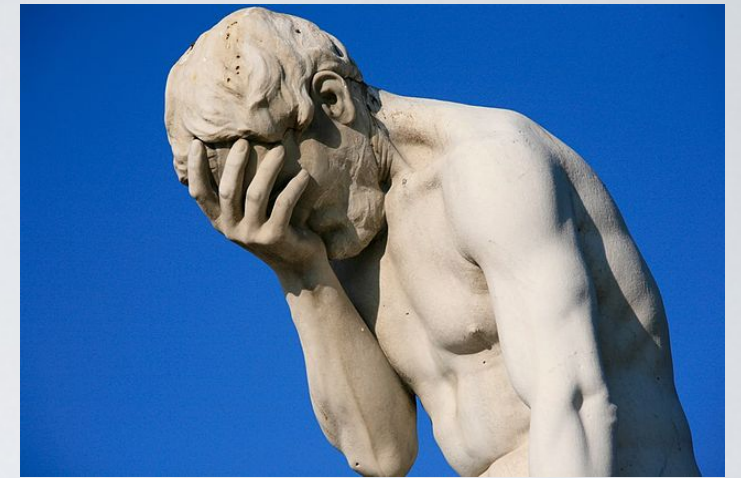
'Serious' Microsoft Office Encryption Flaw Uncovered

 [COMMENTS](#)

By [John E. Dunn](#), IDG News Service
Jan 27, 2005 4:00 PM

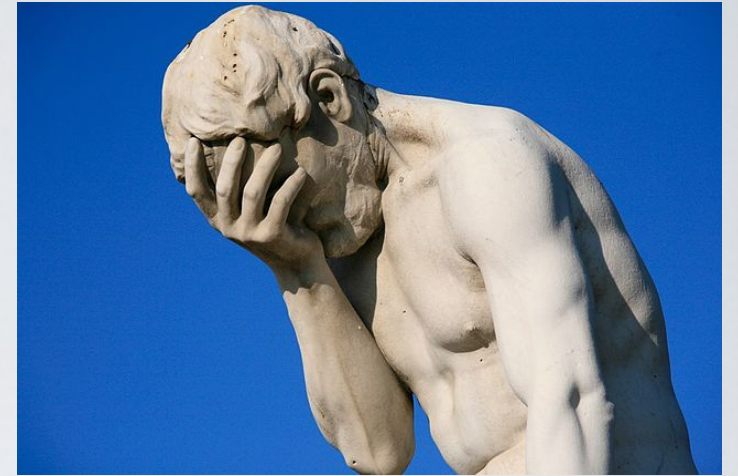
Cryptography expert Phil Zimmermann says he believes a flaw recently discovered in Microsoft Office's Word and Excel encryption is serious and warrants immediate attention.

"I think this is a serious flaw--it is highly exploitable. It is not a theoretical attack," says Zimmermann, referring to a flaw in Microsoft's use of RC4 document encryption unearthed recently by a researcher in Singapore.



MS Word and Excel 2003 used the same key to re-encrypt documents after editing changes

WEP - Wired Equivalent Privacy



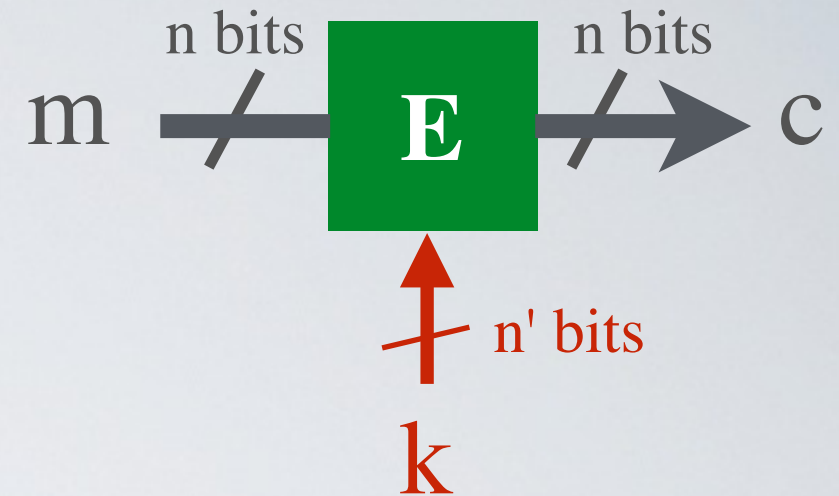
- ➔ A random number IV (24 bits only) transmitted in clear between the clients and the base station

$$\text{RC4_key} = \text{IV} + \text{SSID_password}$$

- ⦿ 50% chance the same IV will be used again after 5000 packets

Block Cipher

Ideal block cipher



- Combines confusion (substitution) and diffusion (permutation)
 - Changing single bit in plaintext block or key results in changes to approximately half the ciphertext bits
- ➡ Completely obscure statistical properties of the original message
- ➡ A known-plaintext attack does not reveal the key

DES - Data Encryption Standard

| | |
|------------|----------------------|
| Block size | 64 bits |
| Key Size | 56 bits |
| Speed | ~ 50 cycles per byte |
| Algorithm | Feistel Network |

Timeline

- **1972** NBS call for proposals
- **1974** IBM Lucifer proposal
analyzed by DOD and enhanced by NSA
- **1976** adopted as standard
- **2004** NIST withdraws the standard

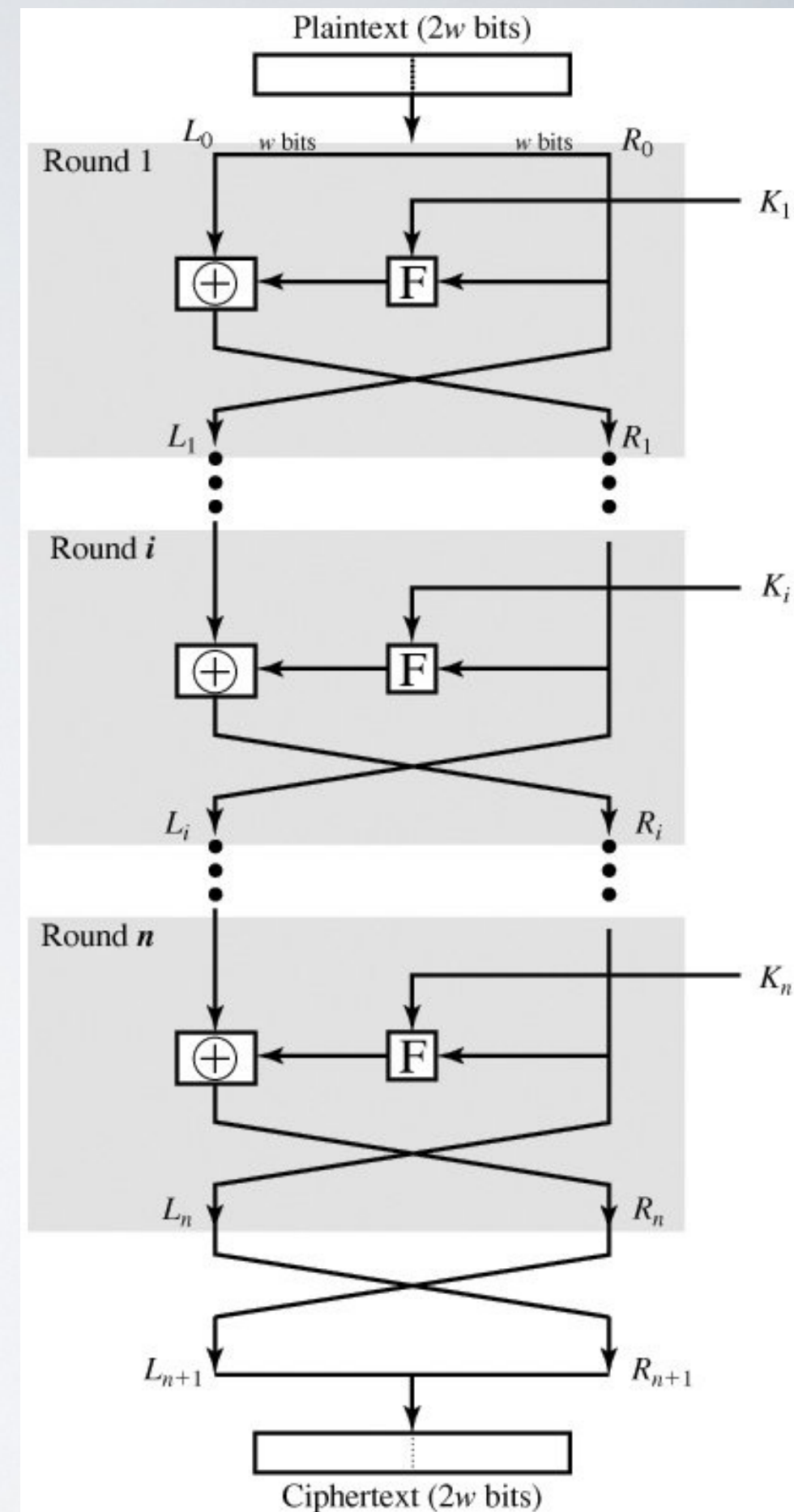
Feistel Network

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F_i(R_{i-1}, k_i)$$

Properties:

- F is an arbitrary function that scrambles the input based on a key
 - F is not necessary invertible
 - A Feistel Network is invertible
- ➡ Achieves confusion and diffusion



Security of DES - DES Challenges (brute force contests)

1998 *Deep Crack*, the EFF's DES cracking machine used 1,856 custom chips

- Speed : matter of days
- Cost : \$250,000

2006 *COPACOBANA*, the COst-optimized Parallel COdeBreaker used 120 FCPGAs

- Speed : less than 24h
- Cost : \$10,000

How about 2DES ?

$$2DES_{k_1, k_2}(m) = E_{k_2}(E_{k_1}(m))$$

Meet-in-the-middle attack - known-plaintext attack

1. Brute force $E_{k_1}(m)$ and save results in a table called TE (2^{56} entries)
 2. Brute force $D_{k_2}(c)$ and save results in a table called TD (2^{56} entries)
 3. Match the two tables together to get the key candidates
- ➡ The more plaintext you know, the lesser key candidates
 - ➡ Effective key-length (entropy) is **57 bits**
 - ➡ This attacks applies to every encryption algorithm used as such

3DES (Triple DES)

$$3DES_{k1,k2,k3}(m) = E_{k3}(D_{k2}(E_{k1}(m)))$$

- ➡ Effective key length (entropy) : 112 bits
- ✓ Very popular, used in PGP, TLS (SSL) ...
- ⦿ But terribly slow

AES - Advanced Encryption Standard

Timeline

- **1996** NIST issues public call for proposal
- **1998** 15 algorithms selected
- **2001** winners were announced

Rijindael by *J. Daemen and V. Rijmen*

| | |
|-------------------------|--|
| Block size | 128 bits |
| Key Size | 128, 192, 256 bits |
| Speed | ~18-20 cycles / byte |
| Mathematical Foundation | Galois Fields |
| Implementation | <ul style="list-style-type: none">• Basic operations : \oplus, $+$, shift• Small code : 98k |

Adopted by the NIST in December 2001

Encryption Modes

a.k.a. how to encrypt long messages

ECB - Electronic Code Book

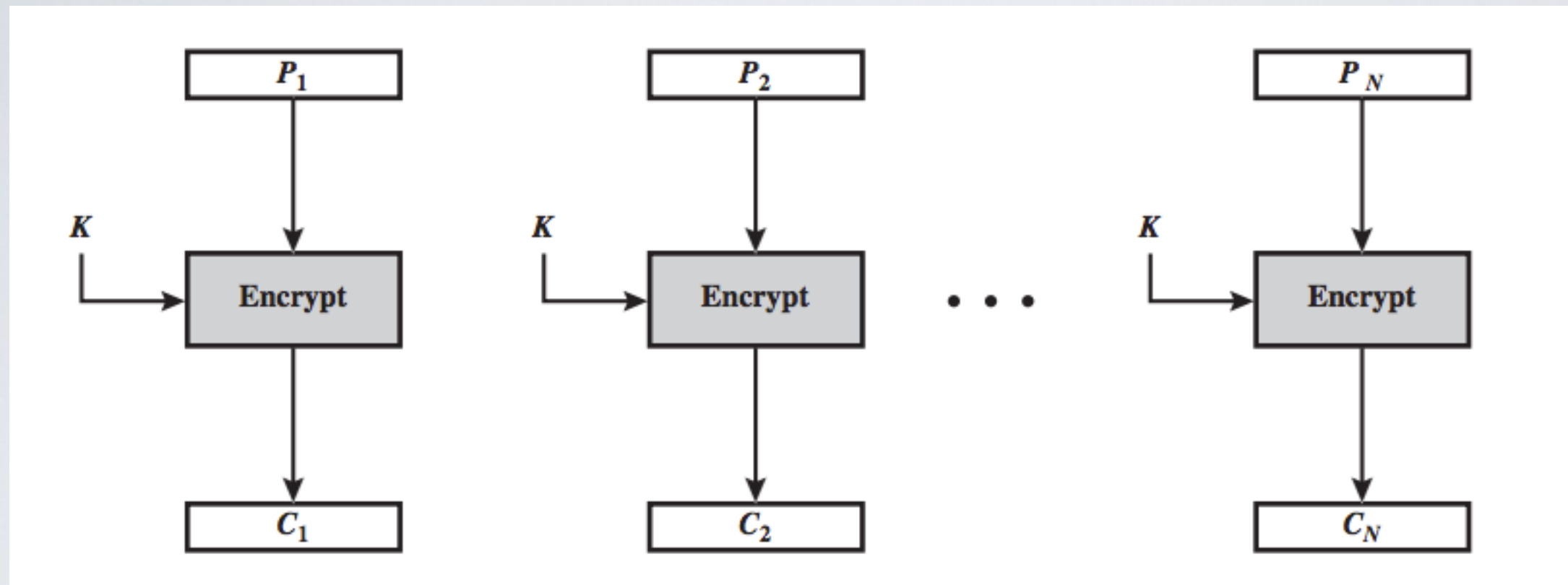
CBC - Cipher Block Chaining

CFB - Cipher Feedback

OFB - Output Feedback

CTR - Counter

ECB - Electronic Code Book



Each plaintext block is encrypted independently with the key

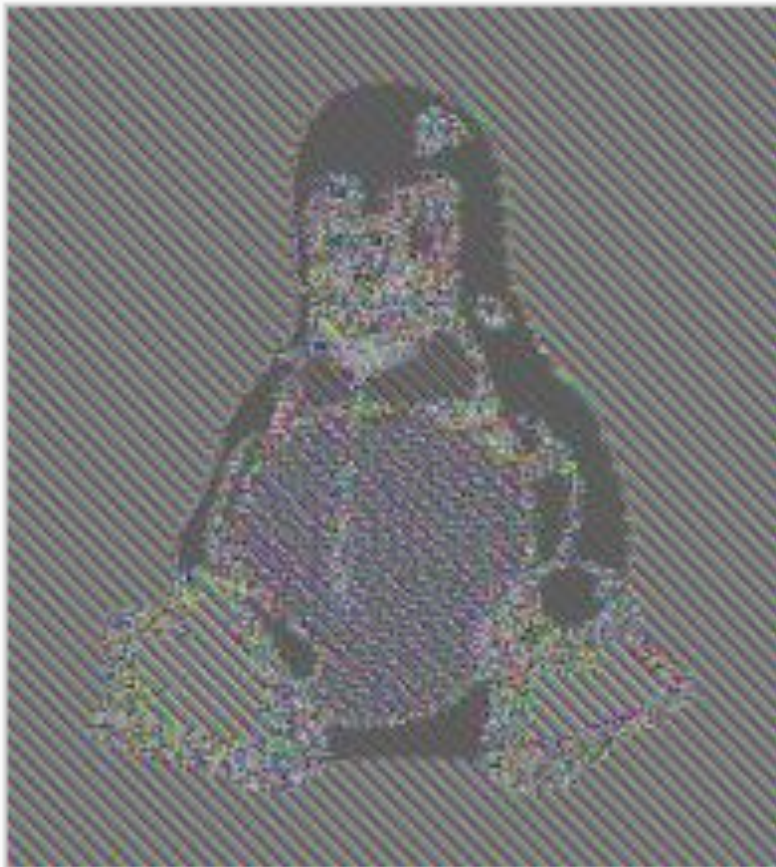
✓ Block can be encrypted in parallel

⦿ The same block is encrypted to the same ciphertext

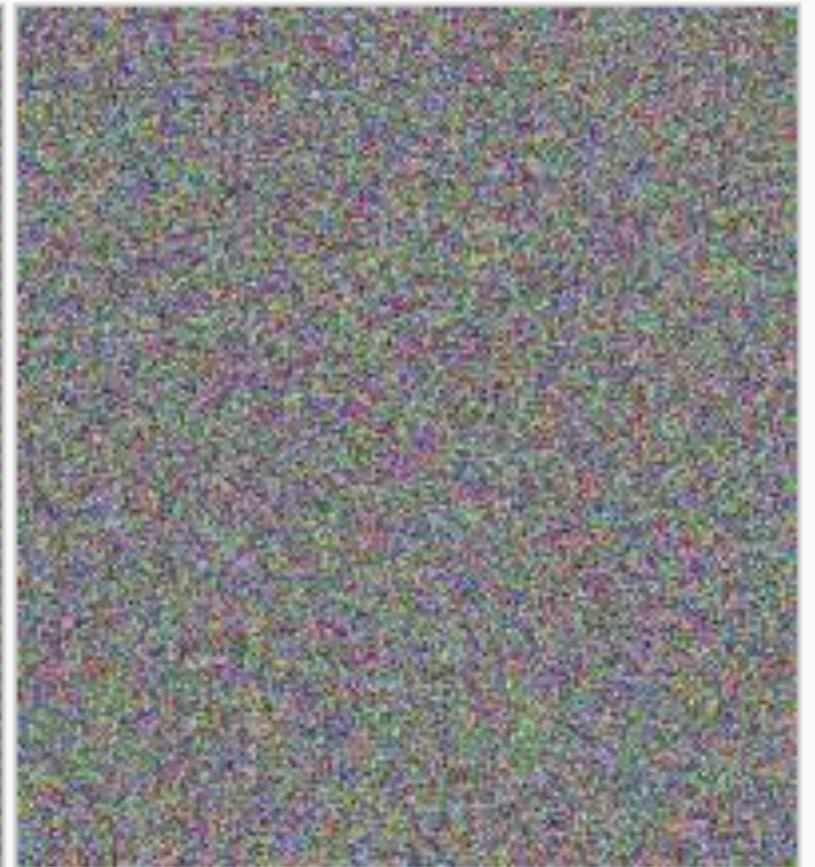
How bad is ECB mode with a large data?



Original image



Encrypted using ECB mode

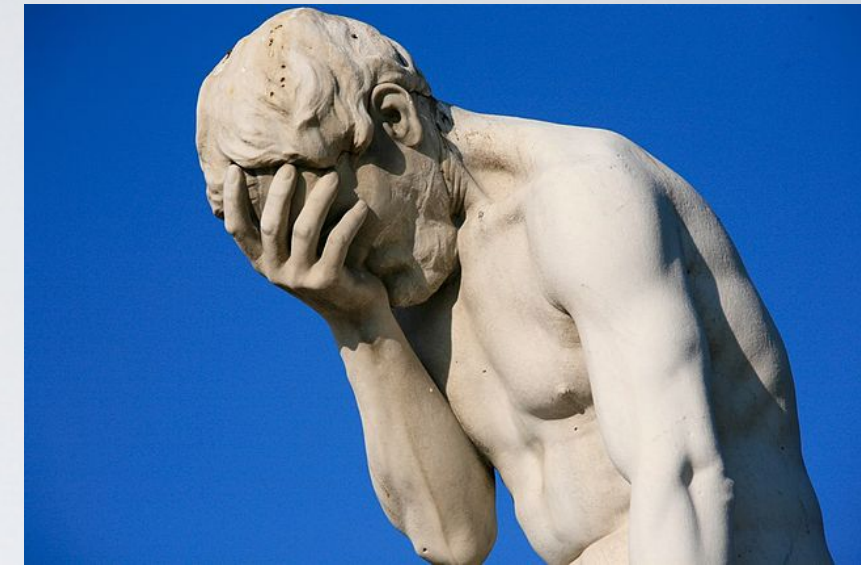


Modes other than ECB result in pseudo-randomness

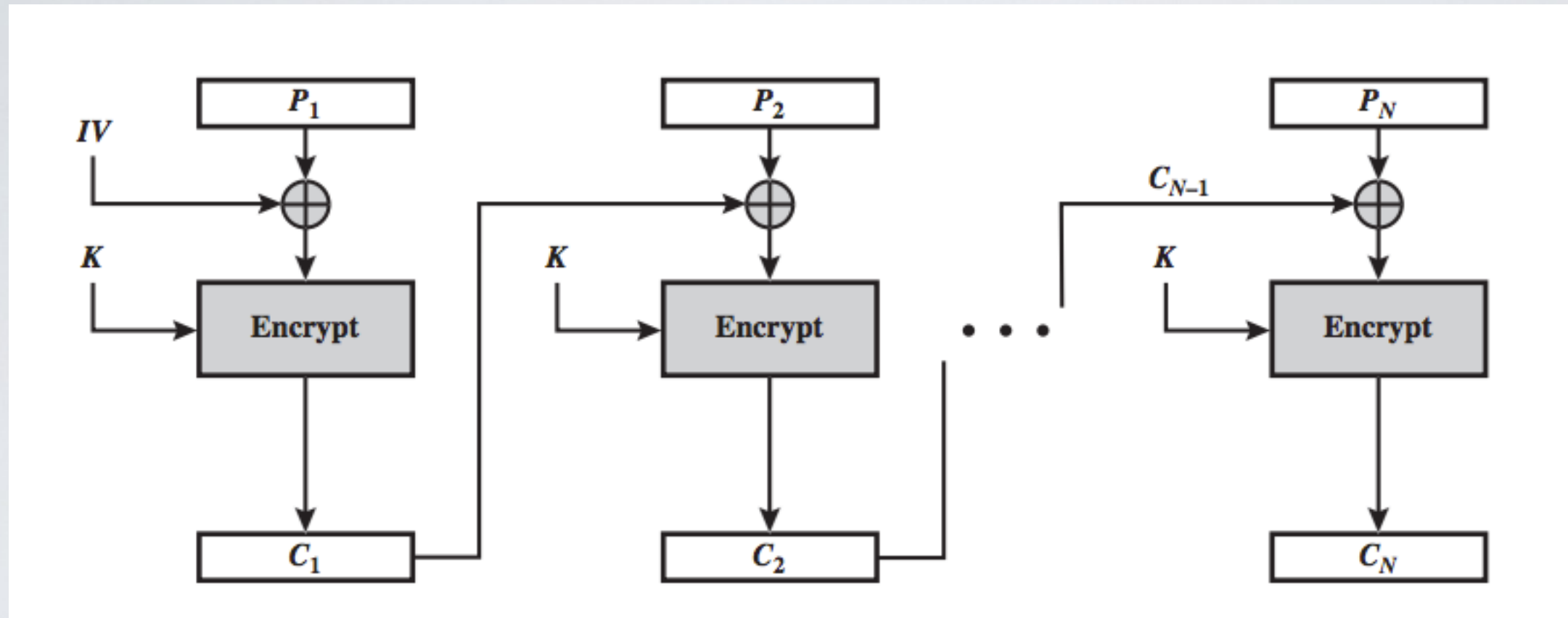
HACKERS RECENTLY LEAKED **153 MILLION** ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS. ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

| USER | PASSWORD | HINT | |
|-------------------|------------------|---|----------------------|
| 4e18acc1ab27a2d6 | | WEATHER VANE SWORD | <input type="text"/> |
| 4e18acc1ab27a2d6 | | | <input type="text"/> |
| 4e18acc1ab27a2d6 | a0a2876eb1ea1fca | NAME 1 | <input type="text"/> |
| 8babbb6299e06eb6d | | DUH | |
| 8babbb6299e06eb6d | a0a2876eb1ea1fca | | <input type="text"/> |
| 8babbb6299e06eb6d | 85e9da81a8a78adc | 57 | |
| 4e18acc1ab27a2d6 | | FAVORITE OF 12 APOSTLES | |
| 1ab29ae86da6e5ca | 7a2d6a0a2876eb1e | WITH YOUR OWN HAND YOU HAVE DONE ALL THIS | |
| a1f9b2b6299e7a2b | e0dec1e6ab797397 | SEXY EARLOBES | <input type="text"/> |
| a1f9b2b6299e7a2b | 617ab0277727ad85 | BEST TOS EPISODE | <input type="text"/> |
| 39738b7adb0b8af7 | 617ab0277727ad85 | SUGARLAND | |
| 1ab29ae86da6e5ca | | NAME + JERSEY # | |
| 877ab7889d3862b1 | | ALPHA | <input type="text"/> |
| 877ab7889d3862b1 | | | <input type="text"/> |
| 877ab7889d3862b1 | | | <input type="text"/> |
| 877ab7889d3862b1 | | OBVIOUS | <input type="text"/> |
| 877ab7889d3862b1 | | MICHAEL JACKSON | <input type="text"/> |
| 38a7c9279codeb44 | 9dca1d79d4dec6d5 | | |
| 38a7c9279codeb44 | 9dca1d79d4dec6d5 | HE DID THE MASH, HE DID THE | <input type="text"/> |
| 38a7c9279codeb44 | | PURLOINED | <input type="text"/> |
| 08ae5745a7b7af7a | 9dca1d79d4dec6d5 | FAV. LATER-3 POKEMON | |

THE GREATEST CROSSWORD PUZZLE
IN THE HISTORY OF THE WORLD



CBC - Cipher Block Chaining



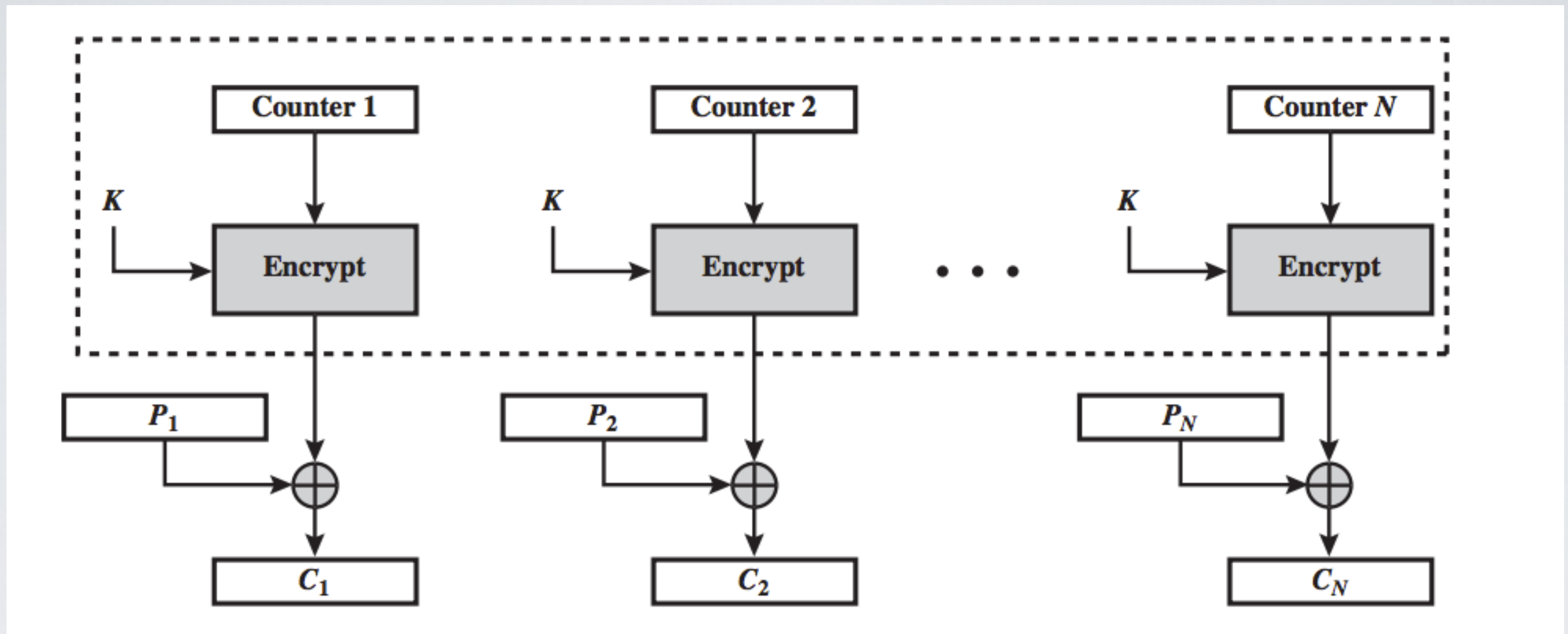
Introduce some randomness using the previous ciphertext block

✓ Repeating plaintext blocks are not exposed in the ciphertext

⦿ No parallelism

➡ The Initialization Vector should not be known by the opponent and must be sent separately (ECB mode for instance)

CTR - Counter



Introduce some randomness using a counter

✓ High entropy and parallelism

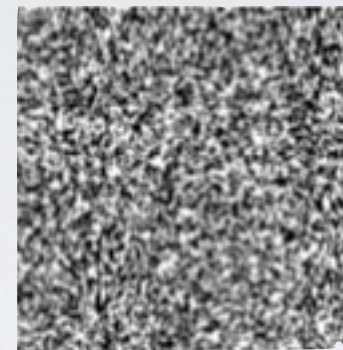
⦿ Sensitive to key-reused attack

➡ Popular usage : IPsec (coming soon in this course)

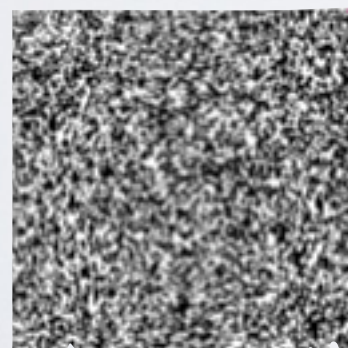
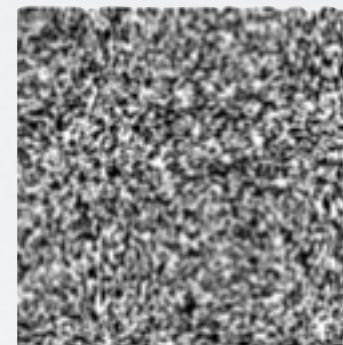
Key-reused attack on CTR



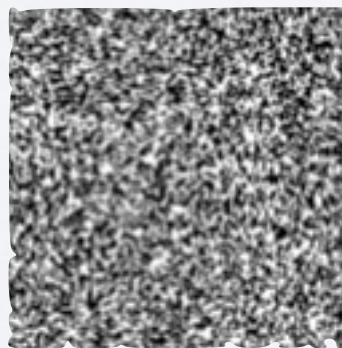
$$\oplus K =$$



$$\oplus K =$$



$$\oplus$$



$$=$$



Stream Cipher vs Block Cipher

| | Stream Cipher | Block Cipher |
|----------|--|---|
| Approach | Encrypt one symbol of plaintext directly into a symbol of ciphertext | Encrypt a group of plaintext symbols as one block |
| Pro | Fast | High diffusion |
| Cons | Low diffusion | Slow |

Stream cipher and block cipher are often used together

- Stream cipher for encrypting large volume of data
- Block cipher for encrypting fresh pseudo-random seeds